ISSN: 2319-0124
josif 2023
15ª JORNADA CIENTÍFICA E TECNOLÓGICA
12º SIMPÓSIO DE PÓS-GRADUAÇÃO
DO IFSULDEMINAS
REALIZAÇÃO INSTITUTO FEDERAL Sul de Minas Gerais
APOIO FAPEMIG CNPq

ESTUDO EXPERIMENTAL DA PROGRAMAÇÃO PARALELA COM MPI NA MULTIPLICAÇÃO DE MATRIZES EM UM CLUSTER COM 8 TVs BOX

Fabiano SANTOS[1]; Gustavo MENDONÇA[2]; Júlio COSTA[3]; Ryan ALVES[4];Victoria MORAES[5]; João P. de T. GOMES[6]

RESUMO

Este trabalho apresenta uma investigação sobre o uso do MPI (*Message Passing Interface*) para realizar processamento paralelo em um código de multiplicação de matrizes. O objetivo principal é medir o tempo de execução em diferentes configurações de *threads* e analisar o impacto no desempenho. Utilizando a função MPI_Wtime() para calcular o tempo, foram realizados experimentos em que o número de *threads* variou de 1 a 32. Os resultados mostraram melhorias significativas no tempo de execução à medida que mais *threads* foram utilizadas, indicando uma utilização mais eficiente dos recursos de processamento. No entanto, observou-se que a partir de um ponto, a adição de novos threads não trouxe benefícios adicionais. Como trabalhos futuros, são sugeridas otimização de algoritmos e experimentos diversos. Este estudo contribui para a compreensão do desempenho do MPI e abre caminho para a utilização mais eficiente do processamento paralelo em aplicações futuras.

Palavras-chave: *OpenMPI*; *Threads*; *Desempenho*; Paralelismo.

1. INTRODUÇÃO

A crescente demanda por processamento de dados complexos e computacionalmente demorados impulsionou o desenvolvimento de técnicas de programação paralela (BRAUN, 2006). A multiplicação de matrizes é uma operação que necessita de grande poder computacional, amplamente utilizada em diversas áreas, como medicina, aprendizado de máquina e processamento de imagens (MACHADO; ZAMITH, 2019). A utilização de clusters, que consistem em várias máquinas trabalhando em conjunto, oferece uma solução escalável para acelerar o processamento de tarefas paralelas. O objetivo principal foi avaliar o desempenho de um algoritmo de multiplicação de matrizes em diferentes configurações de *threads*, buscando identificar o impacto da paralelização no tempo de execução e na eficiência computacional. A multiplicação de matrizes é uma operação altamente paralelizável, uma vez que as operações elementares podem ser realizadas independentemente em subconjuntos dos elementos das matrizes (ANJOS; ITO, 2019). A utilização do padrão de comunicação MPI permite distribuir as tarefas de multiplicação entre os nós

[1] Discente do curso de Bacharelado em Ciência da Computação, IFSULDEMINAS – *Campus* Passos. E-mail: fabiano.santos@alunos.ifsuldeminas.edu.br.
[2] Discente do curso de Bacharelado em Ciência da Computação, IFSULDEMINAS – *Campus* Passos. E-mail: gustavo.mendonca@alunos.ifsuldeminas.edu.br.
[3] Discente do curso de Bacharelado em Ciência da Computação, IFSULDEMINAS – *Campus* Passos. E-mail: julio.costa@alunos.ifsuldeminas.edu.br.
[4] Discente do curso de Bacharelado em Ciência da Computação, IFSULDEMINAS – *Campus* Passos. E-mail: ryan.alves@alunos.ifsuldeminas.edu.br.
[5] Discente do curso de Bacharelado em Ciência da Computação, IFSULDEMINAS – *Campus* Passos. E-mail: victoria.atilio@alunos.ifsuldeminas.edu.br.
[6] Professor, IFSULDEMINAS – *Campus* Passos. E-mail: joao.gomes@ifsuldeminas.edu.br.

do *cluster,* aproveitando a capacidade de processamento distribuído para acelerar o cálculo.

A programação paralela é a divisão de uma determinada aplicação em partes menores, de maneira que essas partes possam ser executadas simultaneamente, em diferentes processadores ou nós de um sistema (RAUBER; RUNGER 2010). Isso permite aproveitar o poder de processamento distribuído e acelerar a execução de tarefas complexas. O padrão de comunicação *Message Passing Interface* (MPI) é uma especificação amplamente adotada para programação paralela em sistemas distribuídos. Ele fornece um conjunto de funções que permitem a troca de mensagens entre os processos em um ambiente de computação paralela, facilitando a comunicação e a sincronização entre eles (REBONATTO, 2004). Por exemplo, a multiplicação de duas matrizes, A e B, resulta em uma matriz C, onde cada elemento C[i][j] é calculado como o produto interno da linha i da matriz A e da coluna j da matriz B. Por esse motivo a multiplicação de matrizes pode ser paralelizada, visto que as operações elementares podem ser divididas e realizadas independentemente em subconjuntos dos elementos das matrizes. Através do MPI, é possível distribuir as tarefas de multiplicação de matrizes entre os nós do *cluster*, aproveitando a capacidade de processamento paralelo e reduzindo o tempo necessário para a conclusão da operação. A paralelização da multiplicação de matrizes permite a execução simultânea de múltiplas operações, resultando em ganhos de desempenho.

2. MATERIAL E MÉTODOS

Para realizar os experimentos e avaliar o desempenho da multiplicação de matrizes com MPI, foi construído um ambiente de simulação que consiste em um *cluster* composto por 8 TVs *box*, totalizando 32 *threads* disponíveis para execução (ou seja, cada nó da *box* com 4 núcleos de processamento). Cada TV *box* atua como um nó de processamento independente, e a comunicação entre os nós é realizada por meio de uma rede local. O *cluster* foi configurado com um Sistema Operacional Linux e dividido em um Servidor (*Server*) e mais sete nós (node1, node2, node3, node4, node5, node6 e node7). As orientações e roteiro de configuração podem ser obtidas em: *https://www.espacomaker-passos.com/laboratorios-e-materias-de-apoio*.

Foi utilizado uma implementação na linguagem C com a adição da biblioteca do MPI. Para a medição do tempo foi utilizada a função “MPI_Wtime”, a qual é incluída na biblioteca do MPI. Isso proporcionou um controle sobre os parâmetros de execução, permitindo a medição precisa do tempo de processamento em diferentes quantidades de *threads.*

3. RESULTADOS E DISCUSSÃO

Para avaliar o desempenho do código, foi realizada uma série de experimentos, variando o número de *threads* utilizados. Cada teste foi repetido três vezes para se obter resultados mais precisos. Os tempos de execução obtidos são apresentados através dos Quadros 1, 2 e 3.

Quadro 1 – Tempos de execução de 1 a 11 *threads*.

Threads	1	2	3	4	5	6	7	8	9	10	11
Tempo 1	28,07	14,21	9,52	7,26	6,51	5,62	5,02	6,07	5,99	6,14	6,35
Tempo 2	28,05	14,14	9,52	7,31	6,54	5,73	5	5,93	6,06	6,2	6,35
Tempo 3	28,05	14,14	9,54	7,29	6,5	5,63	5,07	6	6	6,2	6,36

Quadro 2 – Tempos de execução de 12 a 22 *threads*.

Threads	12	13	14	15	16	17	18	19	20	21	22
Tempo 1	6,67	6,94	7,21	7,56	4,96	5,22	5,59	6,21	6,49	6,38	6,23
Tempo 2	6,59	6,97	7,29	7,62	4,91	5,11	5,57	6,01	6,68	6,24	6,38
Tempo 3	6,62	6,91	7,23	7,5	4,96	5,24	5,51	6,16	6,72	6,14	6,22

Quadro 3 – Tempos de execução de 23 a 32 *threads*.

Threads	23	24	25	26	27	28	29	30	31	32
Tempo 1	6,02	5,64	5,42	5,69	5,25	5,49	4,83	4,8	4,52	3,92
Tempo 2	5,9	6,11	5,51	5,38	5,13	5,52	4,8	4,53	4,73	4,01
Tempo 3	5,85	5,84	5,66	5,03	5,17	5,23	4,87	5,1	4,86	3,95

Na Figura 1 é possível observar um gráfico contendo a média dos tempos de execução de acordo com a quantidade de *threads* utilizados na execução do código.

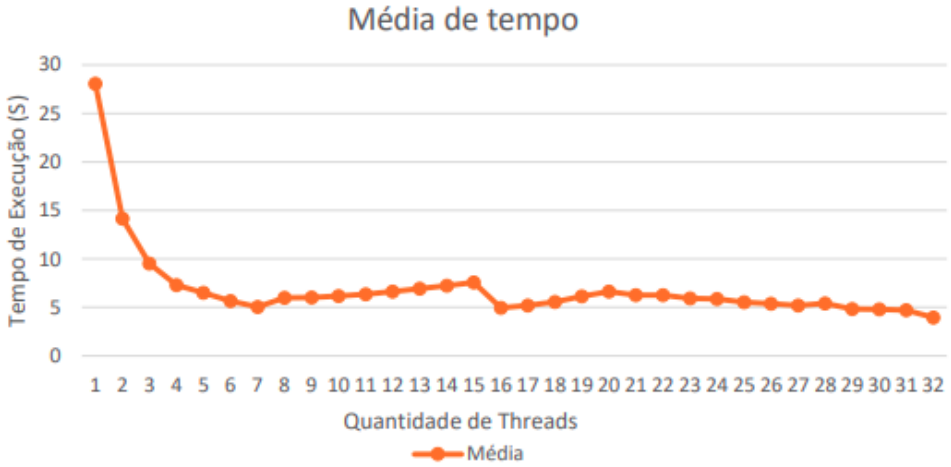

Figura 1 - Média dos tempos de execução

Observando o gráfico, pode-se notar algumas tendências interessantes e características da distribuição de trabalho entre os núcleos, como por exemplo, onde com uma *thread*, o tempo de execução é de 28.1 segundos, ao dobrar a quantidade de *threads*, o tempo é reduzido para 14.2 segundos, o que pode-se considerar uma melhoria significativa em relação à execução sequencial. À medida que o número de *threads* é adicionado, o tempo de execução continua a diminuir. Isso é esperado, uma vez que o processamento paralelo pode dividir a carga de trabalho entre as *threads*, permitindo um processamento mais eficiente. Notavelmente, a partir de oito *threads* (de 32 *threads*), observa-se um tempo de execução constante de aproximadamente 6.00 segundos. Essa observação sugere que, a partir de um ponto, a adição de mais *threads* não resulta em uma melhoria

adicional de eficiência. Isso pode ser devido a fatores como a capacidade de processamento do sistema visto que além disso, o tamanho máximo da matriz testada foi de apenas 835.

4. CONCLUSÃO

Neste trabalho, foi utilizado o MPI para realizar processamento paralelo e medir o tempo de execução de um código de multiplicação de matrizes. Os resultados mostraram que o uso do MPI apresenta melhorias significativas no desempenho de execução, à medida que o número de *threads* aumentava. Observou-se uma diminuição no tempo de execução à medida que as *threads* eram adicionadas ao processo, indicando uma utilização mais eficiente dos recursos de processamento. No entanto, notou-se que a partir de 8 *threads*, novas adições não trouxeram uma melhoria adicional no tempo de execução.

Como trabalhos futuros, pretende-se investigar formas de otimizar o algoritmo utilizado, e explorar outras técnicas de paralelização. Além disso, pretende-se realizar experimentos em diferentes sistemas operacionais, configurações de *hardware* e tamanhos de problema para obtenção de resultados sobre o desempenho do código MPI em diferentes cenários. Contudo, o presente trabalho demonstra o potencial do MPI para melhorar o desempenho por meio do processamento paralelo, onde através de otimizações contínuas e aprimoramentos, é possível alcançar um desempenho ainda melhor com a utilização do MPI.

REFERÊNCIAS

ANJOS, M. A.; ITO, G. C. **Uso de Paralelização na Multiplicação de Matrizes Usando a Biblioteca OpenMP.** In: Universidade Tecnologia Federal do Paraná (UTFPR), 2019. Disponível em: <http://goclasses.sh.utfpr.edu.br/wp-content/uploads/2021/06/PPD-2020-2-Marcos.pdf>. Acesso em: 10/07/2023.

BRAUN, R. L. **Avaliação da implementação de clusters de computadores usando tecnologias livres**. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Feevale, Novembro de 2006. Disponível em: <https://tconline.feevale.br/tc/files/0001_623.doc>. Acesso em: 10/07/2023.

MACHADO, Nickolas R.; ZAMITH, Juliana M.N.S. **Modernização de código: estudo de caso utilizando multiplicação de matriz**. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DO RIO DE JANEIRO (ERAD-RJ), 5. , 2019, Rio de Janeiro. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 43-45. Disponível em: <https://sol.sbc.org.br/index.php/eradrj/article/view/9542/9440>. Acesso em: 09/07/2023.

RAUBER, T.; RUNGER, G. **Parallel Programming for Multicore and Cluster Systems**. Springer, 2010. DOI: https://doi.org/10.1007/978-3-642-37801-0. Acesso em: 09/07/2023.

REBONATTO, M. T. **Introdução a programação paralela com MPI em agregados de computadores (clusters)**. Congresso Brasileiro de Ciência da Computação, Itajaí, 2004, 938 – 955. Itajaí, SC – Brasil, ISSN 1677-2822. Disponível em: <http://www.ufrgs.br/niee/eventos/CBCOMP/2004/html/mini_curso/MiniCurso%2007.pdf>. Acesso em: 09/07/2023.